# Using XML and Java for Astronomical Instrumentation Control

**Troy Ames[a], Lisa Koons[b], Ken Sall[b], Craig Warsaw[b]**

[a]NASA Goddard Space Flight Center
[b]AppNet, Inc.

## ABSTRACT

Traditionally, instrument command and control systems have been highly specialized, consisting mostly of custom code that is difficult to develop, maintain, and extend. Such solutions are initially very costly and are inflexible to subsequent engineering change requests, increasing software maintenance costs. Instrument description is too tightly coupled with details of implementation.

NASA Goddard Space Flight Center is developing a general and highly extensible framework that applies to any kind of instrument that can be controlled by a computer. The software architecture combines the platform independent processing capabilities of Java with the power of the Extensible Markup Language (XML), a human readable and machine understandable way to describe structured data. A key aspect of the object-oriented architecture is software that is driven by an instrument description, written using the Instrument Markup Language (IML). IML is used to describe graphical user interfaces to control and monitor the instrument, command sets and command formats, data streams, and communication mechanisms.

Although the current effort is targeted for the High-resolution Airborne Wideband Camera, a first-light instrument of the Stratospheric Observatory for Infrared Astronomy, the framework is designed to be generic and extensible so that it can be applied to any instrument.

**Keywords:** astronomy, infrared astronomy, XML, instrument: description, telescope: control, SOFIA, NASA: GSFC, software: methodology, Java

# Using XML and Java for Astronomical Instrumentation Control

Troy Ames[a], Lisa Koons[b], Ken Sall[b], Craig Warsaw[b]

[a]NASA Goddard Space Flight Center
[b]AppNet, Inc.

## ABSTRACT

Traditionally, instrument command and control systems have been highly specialized, consisting mostly of custom code that is difficult to develop, maintain, and extend. Such solutions are initially very costly and are inflexible to subsequent engineering change requests, increasing software maintenance costs. Instrument description is too tightly coupled with details of implementation.

NASA Goddard Space Flight Center is developing a general and highly extensible framework that applies to any kind of instrument that can be controlled by a computer. The software architecture combines the platform independent processing capabilities of Java with the power of the Extensible Markup Language (XML), a human readable and machine understandable way to describe structured data. A key aspect of the object-oriented architecture is software that is driven by an instrument description, written using the Instrument Markup Language (IML). IML is used to describe graphical user interfaces to control and monitor the instrument, command sets and command formats, data streams, and communication mechanisms.

Although the current effort is targeted for the High-resolution Airborne Wideband Camera, a first-light instrument of the Stratospheric Observatory for Infrared Astronomy, the framework is designed to be generic and extensible so that it can be applied to any instrument.

**Keywords:** astronomy, infrared astronomy, XML, instrument: description, telescope: control, SOFIA, NASA: GSFC, software: methodology, Java

## 1. BACKGROUND

NASA Goddard Space Flight Center's Instrument Remote Control (IRC) project is an ongoing effort led by the Advanced Architectures and Automation Branch (Code 588). The IRC project supports NASA's mission by defining an adaptive framework that provides robust interactive and distributed control and monitoring of remote instruments. The IRC framework will eventually enable trusted astronomers from around the world to easily access infrared instruments (e.g., telescopes, cameras, and spectrometers) located in remote, inhospitable environments, such as the South Pole, a high Chilean mountaintop, or an airborne observatory aboard a Boeing 747. The IRC framework will ultimately enable astronomers, instrument designers, hardware engineers, and other scientists to define new onboard instruments, to control these instruments remotely, and to monitor vital instrument telemetry over an intranet or, for trusted users, possibly the Internet.

The IRC framework will be applied as an operational solution for the Stratospheric Observatory for Infrared Astronomy (SOFIA) project. , controlling the High-resolution Airborne Wideband Camera (HAWC), the Submillimeter And Far Infrared Experiment (SAFIRE, a spectrometer), and SOFIA's telescope assembly. The IRC architecture was also used to develop the control and data acquisition software for one of the proposed detector concepts for the Spectral and Photometric Imaging REceiver (SPIRE), a focal plane instrument for the European Space Agency's (ESA) Far Infrared Space Telescope (FIRST).

IRC is a platform independent framework, designed to be generic and extensible, so that it can be applied to any instrument capable of being computer controlled. In order to design an extensible and flexible architecture, the established goals of the IRC project are to:

- Provide as much platform independence as possible.

- Create a system that is easy to develop, modify, maintain, and extend.
- Explicitly promote reuse by design and by utilizing emerging technologies that facilitate software reuse.
- Greatly reduce the implementation time for facility instruments, which are reliable, robust, state-of-the-art instruments that are easily used by scientists other than the instrument's designers.
- Clearly define the interface between hardware and software engineers.
- Facilitate multiple iterations of the instrument description during design and implementation by means of a software architecture that is readily adaptable to such changes.
- Cleanly separate implementation from description.

## 2. INSTRUMENT MARKUP LANGUAGE

The Instrument Markup Language (IML) has been created based on experience gained by focusing on the astronomy domain (and infrared instruments in particular). This enables the development of a general and extensible framework for instrument control. IML is a vocabulary based on a W3C standard, the Extensible Markup Language (XML), currently defined using a Document Type Definition (DTD). A DTD enables the XML parser to validate the XML files, thereby guaranteeing the instrument description is complete and correct according to the content constraints the developers have imposed. A key aspect of the object-oriented architecture, implemented in Java, is that the software is driven by the IML instrument description. The attributes of an instrument that can be described by IML include:

- Instrument subsystems
- Logical command set
- Command arguments and units
- Argument data types and valid values/ranges
- Command formats
- Logical data streams (e.g., science data, housekeeping, command responses)
- Data field data types
- Data formats
- Communication mechanisms
- Documentation

Although at this stage in the evolution of IML it is the software engineers who are writing the descriptions, the IRC team envisions obtaining the participation of hardware engineers in the task. Not only do hardware engineers best know the instrument details, but they traditionally provide significant contributions to formal Interface Control Documents (ICDs). NASA/GSFC believes that IML documents can serve a similar role – that is, communicating the intricate details of an instrument's operation – in a much more structured, human readable, and easily navigated way (e.g., collapsing and expanding nodes that correspond to different sections of interest). Promoting IML (or some dialect thereof) for use as an ICD will require providing hardware engineers with a reasonable development environment that includes tools that hide XML details. By utilizing an XML editor with a sufficiently self explanatory IML DTD, hardware engineers can be guided through the process of instrument specification by noting what elements are applicable in a certain context based on the DTD's content model. The long-term vision includes providing an Instrument Description Wizard that will further hide the XML details with an interview-style interface.

Although IML is currently applied to astronomical instruments, the key aspects of our approach to instrument description and control apply to many domains, from medical instruments (e.g., microscopes) to printing presses to machine assembly lines. Due to the extensible nature of XML, we can easily imagine extensions to IML for various domains, such as the Astronomical Instrument Markup Language (AIML). At this point, there is no separate AIML DTD – the generic IML DTD provides all of the necessary functionality. A separate AIML DTD will be created when (and if) needs are identified that are not part of the general instrument description solution. The IML (or AIML)

DTD can be extended to cover new instruments or to accommodate changing constraints of existing instruments, often without requiring changes to previously written instrument descriptions.

## 3. INSTRUMENT REMOTE CONTROL ARCHITECTURE

The Instrument Remote Control framework is implemented entirely in Java. Figure 1 shows the high level architecture of the IRC framework. The IML file, read at run time, drives the behavior of many general Java objects.

By default, a Graphical User Interface (GUI) is automatically created by reading in the IML instrument description. This default GUI provides the means to issue all of an instrument's (and an instrument's subsystem's) commands. Since the IML file describes all of the arguments (including the argument's data types and the argument's valid values), the GUI can present a command window that enables a user to issue valid commands.

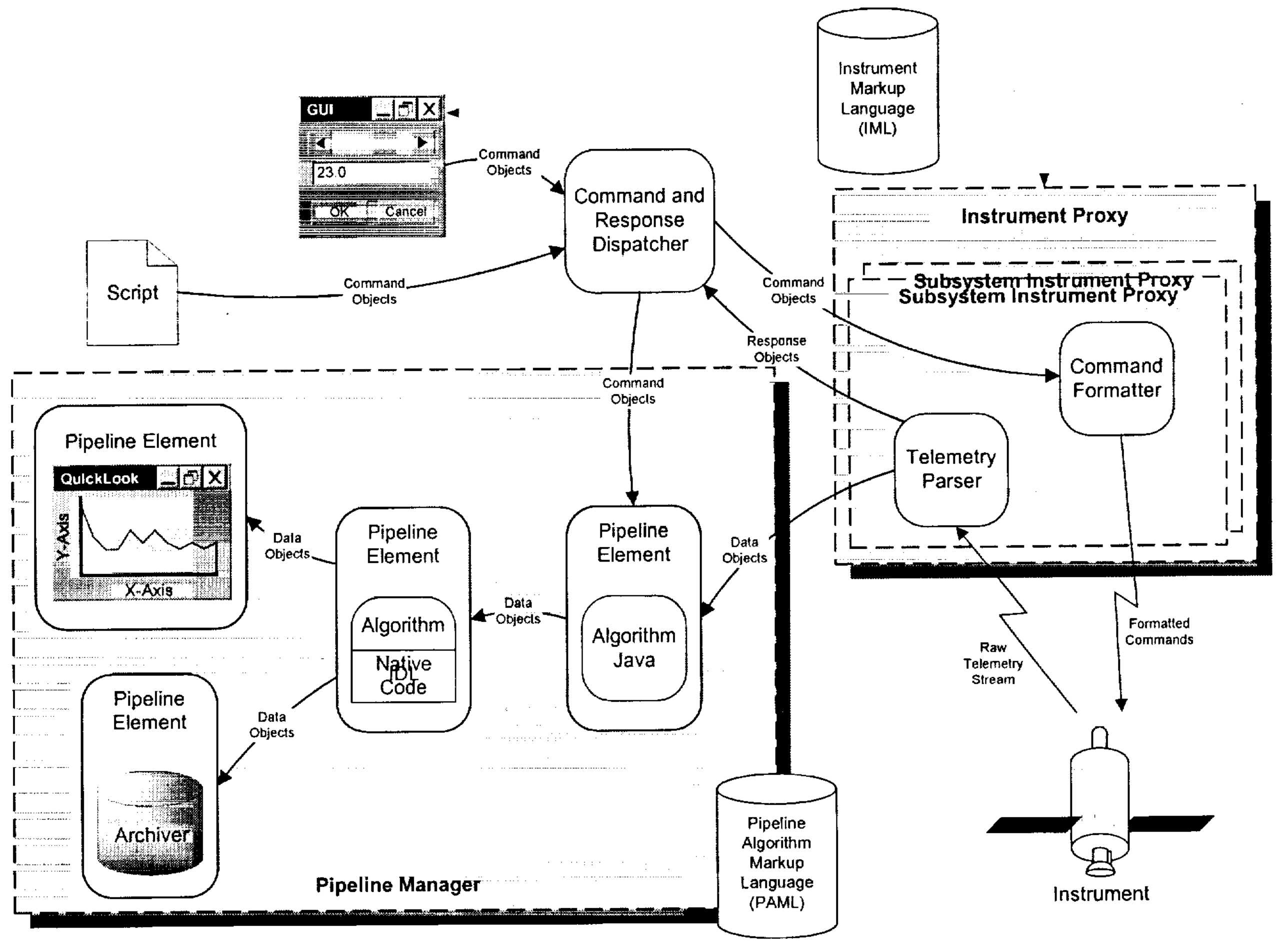


**Figure 1: High Level IRC Framework Architecture**

The Instrument Proxy object creates objects that know how to communicate directly with the instrument. The IML instrument description specifies the communication mechanism (e.g., TCP/IP, GPIB) and the formatting rules for commands. Actually, the IML instrument description can describe a hierarchy of sub-instruments. Each subsystem (sub-instrument) may have a different communication mechanism or protocol. For example, one subsystem may have a TCP/IP interface with binary commands and another subsystem may have an RS232 interface with ASCII commands. Each subsystem in the IML instrument description will cause a separate subsystem Instrument Proxy to be created. Each Instrument Proxy will receive command objects, format them using the rules specified in the IML file (using a generic Command Formatter), and then send them to the actual instrument.

The Instrument Proxy will also create an instance of a Telemetry Parser for each instrument port that emits telemetry as defined in the IML file. The parsing rules described in the IML file define how raw data is parsed into Data Objects (see Section 5: Data Analysis Pipeline).

Many of the generic framework objects allow instrument specific delegates to be plugged in that are specified in the IML instrument description. These delegates can be used to override generic behavior. For example, the SPIRE instrument defined six telemetry streams among four subsystems. The formats of five of the telemetry streams could be described such that the generic telemetry parsing engines could parse those streams sufficiently, resulting in five instances of the generic Telemetry Parser objects. However, the generic parsing engine could not handle the science data stream. The format of the data was extremely complex and the software had to handle data rates of up to 15 MB per second. A highly optimized Java parsing delegate was implemented and tuned to the demands of the SPIRE science data. The class of the delegate was specified in the IML instrument description. At runtime, the Instrument Proxy created an instance of this delegate, and plugged it into the science data port's Telemetry Parser. Java's ability to load classes dynamically makes it trivial to enable the generic framework to perform instrument-specific operations without prior knowledge of the delegate classes.

The flow of Command and Data Objects through the system is facilitated by using a Publish/Subscribe pattern. Subscribers register with publishers for objects that they are interested in. They can register for all objects published, or only for objects that match particular criteria. This facilitates a dynamic and distributed flow of control. Instrument proxies register their Command Formatters as subscribers of commands from a singleton Command Dispatcher. GUIs can come and go based on user demands, and they can even reside on remote machines. When a GUI is created, it notifies the Command Dispatcher, which in turn registers as a subscriber to commands that the GUI publishes. As a result, Command Formatters are unaffected by the transient nature of the GUIs.

## 4. IML EXAMPLES

This section includes some examples that demonstrate how a typical instrument is described using IML. Note that much of the detail is omitted from these examples for presentation purposes. Consider an instrument named HAWC that contains four subsystems: Detector, ADR, Opto, and Telescope.

```
<Instrument id="HAWC">
    <Instrument id="Detector"> …
    <Instrument id="ADR"> …
    <Instrument id="Opto"> …
    <Instrument id="Telescope">
        <Port name="Telescope" function="command" dataType="ASCII"
              hostName="testmachine2.nasa.gov" number="6554"
              type="gov.nasa.gsfc.irc.port.TcpIoPort" serverPort="true">
            <!-- Commands and formats go here -->
        </Port>
    </Instrument>
</Instrument>
```

**Figure 2: IML description of instrument and subsystems**

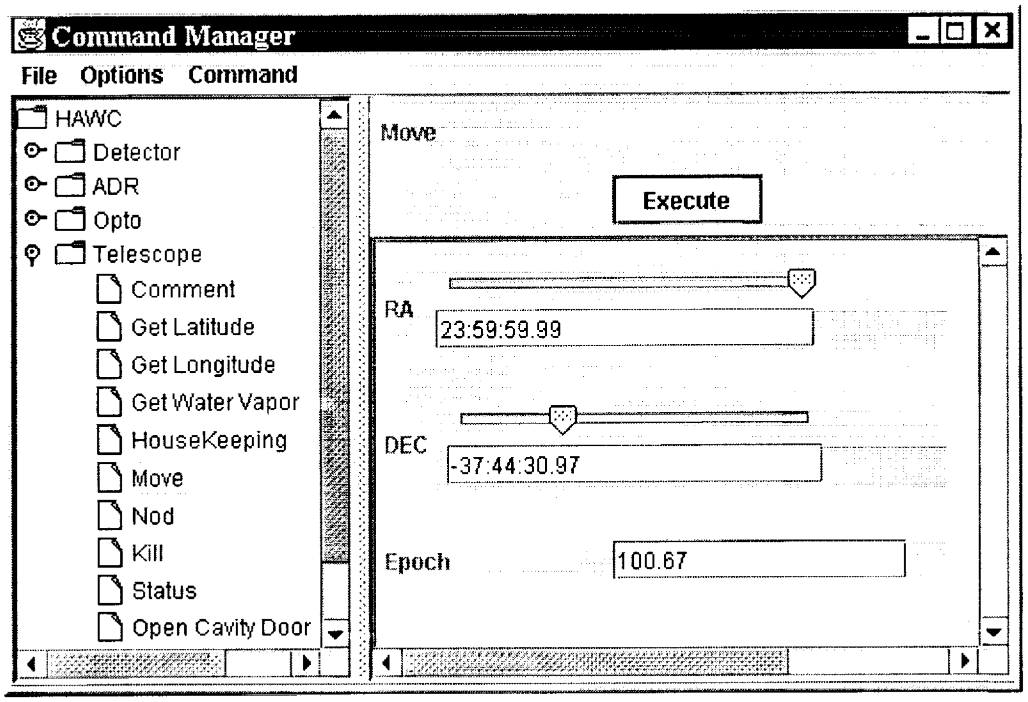
A subsystem can have multiple ports, for example there may be a need for a commanding port and a telemetry port. Figure 2 defines that the Telescope subsystem has a single TCP port for commanding and by examining the port element we see that commands are ASCII. In addition to a TCP port, the software supports, but is not limited to, several other communication mechanisms such as RS232 and DMA.

Now let's examine a couple of the Telescope subsystem commands. The Move Command has three Arguments, two are required and one is optional (see Figure 3). The Epoch argument is specified as a float. Since the argument is not constrained, a textfield component appears in the GUI (see Figure 4).

The GUI ensures that the user input is of the correct type, and if not, an error dialog is displayed. The RA and DEC arguments are specified to be of a special type called Sexagesimal. This example illustrates how a new data type can be used for a command argument. Furthermore, ValidRanges are supplied causing the GUI to automatically include a slider (scale) component that enforces the constraint.

```
<Command name="Move">
    <Argument name="RA" required="true"
              type="gov.nasa.gsfc.irc.datatypes.Sexagesimal">
        <ValidRange low="00:00:00.0" high="23:59:59.99" />
    </Argument>
    <Argument name="DEC" required="true"
              type="gov.nasa.gsfc.irc.datatypes.Sexagesimal" >
        <ValidRange low="-89:59:59.99" high="89:59:59.99" />
    </Argument>
    <Argument name="Epoch" type="java.lang.Float" required="false" />
</Command>
```

**Figure 3: IML description of telescope MOVE command**



RA 23:59:59.99

DEC -37:44:30.97

Epoch 100.67

**Figure 4: Generated GUI**

```
<RecordFormat name="Move" ordered="true" terminator="&#10"
attributeSeparator=" ">
    <Format name="Command" format="MOVE" ordered="true" />
    <Format name="RA" format="%s" ordered="false" header="RA=" />
    <Format name="DEC" format="%s" ordered="false" header="DEC=" />
    <Format name="Epoch" format="%.1f" ordered="false" header="EPOCH=" />
</RecordFormat>
```

**Figure 5: IML description of telescope MOVE command format**

Each command sent to an instrument has an associated format. Based upon the Command and RecordFormat (see Figure 5) in the above Telescope subsystem description, a Move Command sent to the Telescope subsystem looks like this:

```
MOVE RA=23:59:59.99 DEC=-37:44:30.97 EPOCH=100.67 <cr>
```

While there are many advantages to the use of IML, one of the most significant is the ability to defer some of the hardware implementation details as long as necessary during the development period. Software often needs to be developed in parallel with the hardware it is to control. Since hardware engineers may need to change various details as their subsystems are integrated, or as new hardware components with different characteristics are manufactured, it is crucial that the software architecture provide a degree of separation between the objects that represent the system and the hardware nuts-and-bolts.

```
<Command name="Nod">
    <Argument name="Beam" type="java.lang.String" required="true">
        <ValidList name="Beam">
            <Choice>left</Choice>
            <Choice>right</Choice>
            <Choice>switch</Choice>
        </ValidList>
    </Argument>
</Command>
```

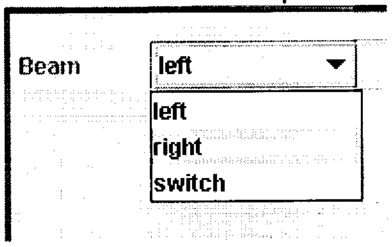


**Figure 6: IML description of telescope NOD command**

IML enables iterative development because the instrument description is parsed at runtime. A concrete example of this is the ability to dynamically alter the graphical user interface to reflect a different hardware interface. For example, the GUI automatically reflects the list of valid choices, specified in IML, as shown in Figure 6. Similarly, suppose the hardware engineer decided that he wants to make available a new Telescope Command. He could simply define his new command and its format in the IML file and the next time the application is run, the new Command would appear in the GUI. Once again, no new code and no recompile would be necessary.

## 5. DATA ANALYSIS PIPELINE

After parsing a raw telemetry stream, the Telemetry Parser publishes the telemetry as Data Objects. Any subscriber in the system can register to receive these Data Objects; however, the IRC framework includes a Data Analysis Pipeline to facilitate the processing of this data. This real-time pipeline is composed of Pipeline Elements that are linked together. Pipeline Elements can be:

- General purpose data processing algorithms, such as parameterized algorithms that apply a scale factor or polynomial function to selected input data
- Instrument specific data processing algorithms
- Archivers and archive readers
- Data visualizations
- Data analysis scripts for autonomous commanding

Pipeline Elements are described using the Pipeline Algorithm Markup Language (PAML), an extension of IML (using the built-in extension capabilities of XML DTDs). With PAML, you must describe the following attributes of a Pipeline Element:

- Implementation class – the Java class that implements the Pipeline Element
- Outputs – identical to the description of an instrument's logical telemetry stream (without formatting information)
- Inputs – a set of characteristics that valid inputs must have
- Properties – a set of attributes that can be used to configure the Pipeline Element

Using the PAML description, IRC provides two mechanisms (GUI or script) for connecting Pipeline Elements together and setting Pipeline Element properties. Since the inputs and outputs are described, the IRC framework will only allow valid connections. Pipeline Elements can be added, removed, or configured while data is flowing through the pipeline.

IRC provides many general purpose algorithms, and attempts to make it easy to develop instrument specific algorithms. Taking advantage of Java's dynamic class loading, the IRC framework doesn't have to know about the algorithm implementation class beforehand. By referencing the location of the Java byte code in the PAML file, the pipeline is able to create instances of the algorithm as needed by the current pipeline configuration. Also, by using the Java Native Interface (JNI), algorithms can be implemented in any native language such as C, C++, or FORTRAN. Conceivably, algorithms can be implemented in IDL as well.

Archivers are a type of Pipeline Element that can be attached to the output of any other Pipeline Element to create a persistent copy of the output of that Pipeline Element. An Archive Reader Pipeline Element can then be used to read the archive. In Figure 7, Pipeline Element B cannot tell whether it is being run with real-time or archived data.

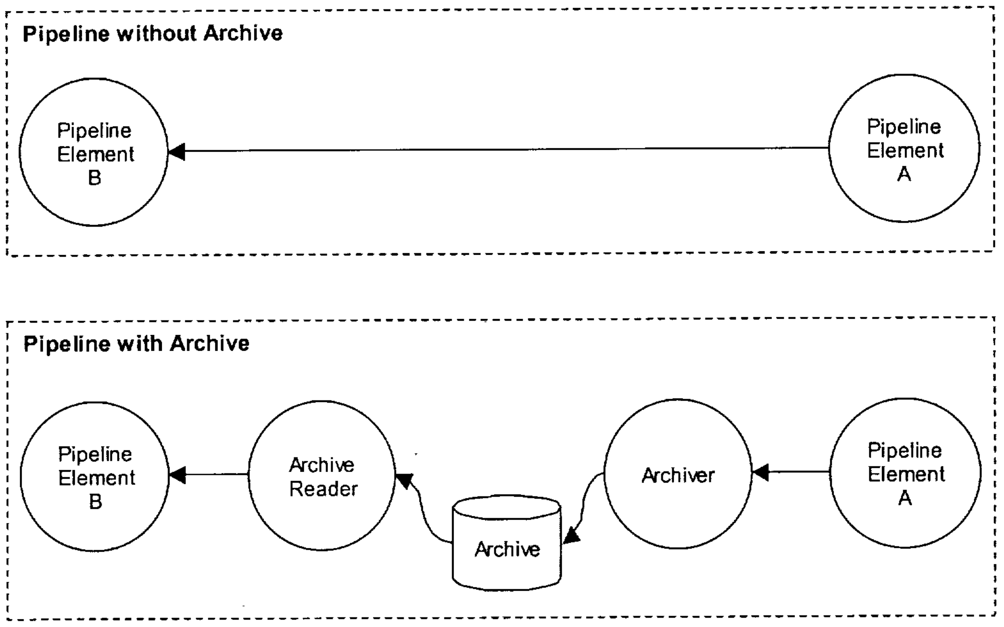


**Figure 7: Adding Archives to a Pipeline**

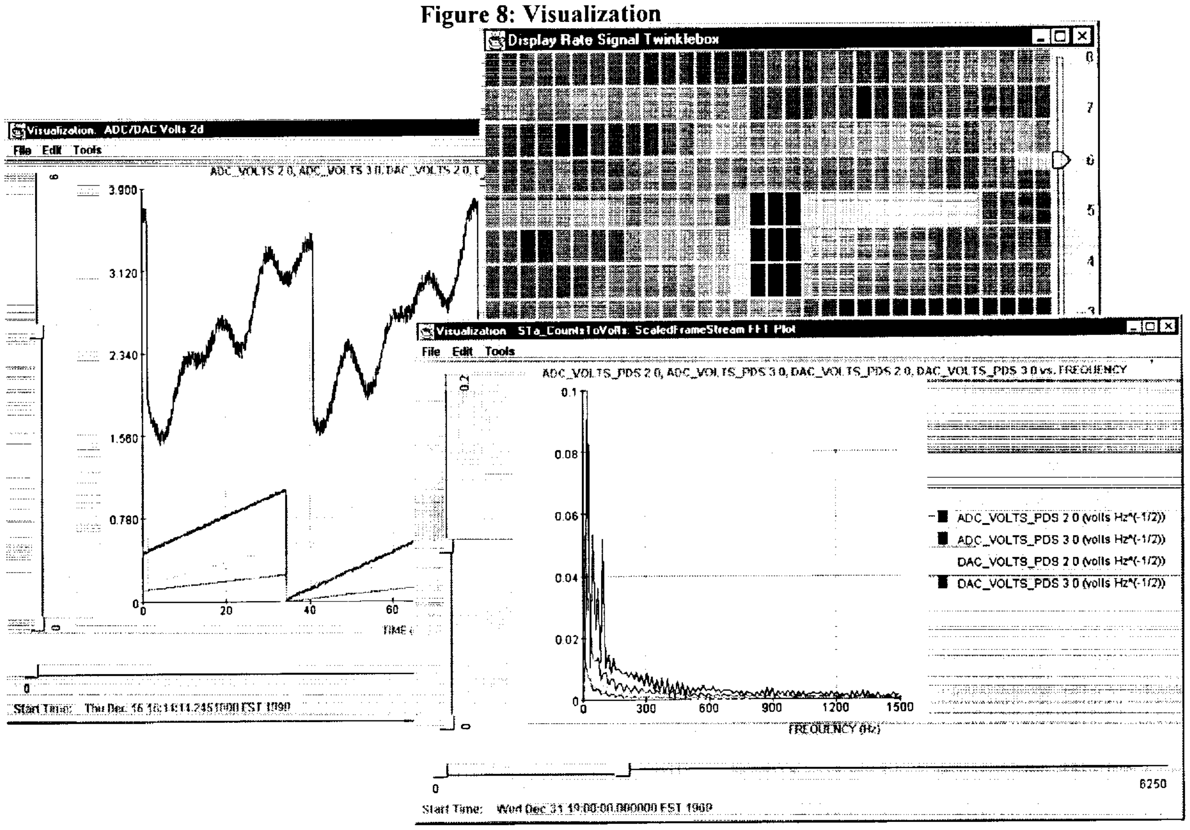
## 6. DATA VISUALIZATIONS

The IRC software provides several visualizations based upon the VisAGE (Visual Analysis Graphical Environment) architecture. VisAGE is another joint-GSFC/AppNet effort providing a distributed, platform independent, multimedia, visualization development environment which takes advantage of JavaBeans, Java2D, and Java3D technologies. The IRC visualizations have been enhanced for efficiency, providing a robust environment to visualize data that must be displayed at high speeds. Notable features include anti-aliasing, logarithmic scales, snapshot printing and saving, and statistics calculation and display.

Visualizations fit into the overall architecture as special purpose Pipeline Elements. This design affords the IRC software some important flexibility. Since a visualization is a Pipeline Element, a visualization instance may be placed anywhere within the pipeline. This allows a user to view raw data early in the pipeline, or the results of complex calculations performed by series of pipeline elements. Furthermore, the user isn't restricted to a single visualization. A user may place many visualizations within the pipeline, providing the ability to track data at many positions within the pipeline simultaneously. This implementation also allows a visualization to publish information, such as statistical or snapshot data, to other parts of the pipeline for archiving or further processing.

The current architecture provides a generic way to select and organize streams or channels of data from upstream Pipeline Elements. Information can then be forwarded to visualizations in a format specialized for display purposes. This facilitates the ability to easily plug new visualizations into the current architecture by allowing them to concentrate on the display of data, rather than its organization. It also accommodates the ability to switch between related visualizations quickly and easily. This design may be expanded to provide wrappers that format data for viewing with third party visualization packages.

**Figure 8: Visualization**


Display Rate Signal Twinklebox
Visualization. ADC/DAC Volts 2d
File Edit Tools
3.900
3.120
2.340
1.560
0.780
0
20
40
60
TIME
Visualization 51a_CountsToVolts: ScaledFrameStream FFT Plot
File Edit Tools
ADC_VOLTS_PDS 2 0, ADC_VOLTS_PDS 3 0, DAC_VOLTS_PDS 2 0, DAC_VOLTS_PDS 3 0 vs. FREQUENCY
0.1
0.08
0.06
0.04
0.02
0
300
600
900
1200
1500
FREQUENCY (Hz)
ADC_VOLTS_PDS 2 0 (volts Hz^(-1/2))
ADC_VOLTS_PDS 3 0 (volts Hz^(-1/2))
DAC_VOLTS_PDS 2 0 (volts Hz^(-1/2))
DAC_VOLTS_PDS 3 0 (volts Hz^(-1/2))
6250
Start Time: Wed Dec 31 19:00:00.000000 EST 1969

**Examples**

## 7. SCRIPTING

The ability to script the instrument control software is an important feature of the IRC framework because it provides the scientist a way to sequence common tasks. Currently, scripts must be written in JPython; however, the IRC architecture allows for future support of other scripting languages. JPython is a Java implementation of Python, an interpreted, object-oriented programming language. JPython and Python are free and the source code is available under an Open Source license.

```
# File: powerup_detector.py
# Initialize the MkII electronics at detector power-up

from gov.nasa.gsfc.irc.datatypes import BitArray

sendCommand(SPIRE_DETECTOR, "Stop Data")
sendCommand(SPIRE_DETECTOR, "Reset Time Counter")
sendCommand(SPIRE_DETECTOR, "Reset Command Counter")
callProcedure("Initialize Row Address Lookup Table")
callProcedure("Initialize Parameter RAM Lookup Table")
callProcedure("Setup FFCP Frame", BitArray("0001"), 8, 100)
```

**Figure 9: Simple Script - Powerup Detector**

Simple scripts that string together a set of instrument commands can be written easily. Such a script is shown in Figure 9. Easy mechanisms are included for prompting the user for input. Support for looping and control flow is implicit in the scripting language. Scripts can add, remove, or configure Pipeline Elements. Using more advanced capabilities of JPython, scripts can be written that extend the IRC framework in interesting ways. JPython has full access to all Java packages and Python modules. Script objects can extend Java objects (and vice versa). These features have been used to create scripts that implement pipeline algorithms, insert themselves into the pipeline, issue commands based on the analysis of incoming data, and then remove themselves from the pipeline.

To make scripts available to the system, a fragment of IML must be written that describes the script, the script arguments including data types and valid values, and any documentation for the script. This IML fragment can be added to a library of scripts or to the description of a subsystem to make the script appear as a primitive command to the user. A script with associated IML fragment is shown in Figure 10: Setup FFCP Frame Script and Figure 11: Setup FFCP Frame IML Description.

```
# This procedure sends the setup FFCP command and also sets the number
# of words for the given frame setup.

sendCommand(SPIRE_DETECTOR, "Setup FFCP", mask, (numRows-1), freq)

hkWords = 4        # extra housekeeping words added to each frame

# calculate the number of columns in the BitArray mask
numCols = 0
for i in range(mask.length()):
 if mask.get(i):
     numCols = numCols + 1

# calculate the number of words in the frame
numWords = ((numCols * numRows) * 2) + numRows + hkWords

sendCommand(SPIRE_DETECTOR,
           "Set Number of Words per Frame to DIO Interface Board",
           numWords)
```

**Figure 10: Setup FFCP Frame Script**

```
<CommandProcedure name="Setup FFCP Frame"  abbrev="setupFrame"
                  language="Python" file="detector/setupFrame.py" >
    <Argument name="Column Mask"
type="gov.nasa.gsfc.irc.datatypes.BitArray"
              default="0001" abbrev="mask">
        <ValidBitRange high="3" /> <!-- 4 bits -->
    </Argument>

    <Argument name="Number of Rows" type="java.lang.Integer"
              default="8" abbrev="numRows">
        <ValidRange low="1" high="32" />
    </Argument>

    <Argument name="Row Select Frequency"
              type="gov.nasa.gsfc.irc.datatypes.InverseHertz"
              default="160" abbrev="freq" unit="kHz">
        <ValidRange low="5" high="4095" /> <!-- 12 bits -->
    </Argument>
</CommandProcedure>
```

**Figure 11: Setup FFCP Frame IML Description**

## 8. THE VISION: FUTURE OF IRC

The IRC framework supports instrument development from early design through operations and maintenance to minimize software development time, minimize development costs, maximize reuse of software components, and maximize flexibility of instrument architectures.

To minimize software development time and to accommodate new instruments, the IRC framework will make it easy to generate large portions of the instrument control application simply from a structured description of the instrument. The description of the instrument will be used to automatically create custom graphical user interfaces, software component interfaces, specific component configuration, and documentation. Software tools will be created to help instrument designers develop descriptions of instruments. Thus, as new instruments are added to a system, or as requirements and specifications of existing instruments are modified, the effort to adapt the software to these changes is incremental, rather than major.

The IRC framework that we envision can be applied to any or all phases of the science life cycle to maximize the science potential of the missions. An important enhancement to the IRC framework will be the capability to quickly develop simulations that accurately model instrument operations to whatever degree of fidelity is deemed necessary. Simulations will enable many activities to be performed long before the instrument development is completed. It will enable instrument designers to develop, validate, and modify designs quickly, efficiently, and very effectively. Scientists will be able to begin science planning and data analysis algorithm development; data archival, retrieval, and publication scenarios can be worked out; and support staff can begin training very early in the program for instrument operations. The underlying open architectural software infrastructure that we envision can be used to assemble and synthesize single and multiple discipline behavior models into instrument simulations to facilitate the rapid design and simulation of next generation instrument designs and operations.

The IRC framework that we are planning will maximize the ability to incorporate emerging technologies. The design supports a high degree of configurability, to be tuned for specific observatories or other factors. Processes can be run on a single computer or multiple heterogeneous computers, ranging from small, low cost hardware components to high-end workstations. Some processes can be run either at the observatory or remotely over the Internet (or both). This provides an instrument development team the flexibility to use the hardware components that best fit the operating environment and instrument requirements. The framework will support the cross-platform migration of functions and reconfiguration if these requirements change. This flexibility enables a new design in which small embedded software components are placed at the point of origin of the generated data (smart sensors) and at the point of device control (smart actuators). The envisioned configurable software framework will enable these software solutions to be easily developed, enhanced, maintained, and reused for different devices, different instruments, and different domains.

To achieve "Smart" instruments and simulations, we envision incorporating advanced technology in the near future. The IRC framework will allow the seamless integration of technologies such as agents, model-based reasoning, genetic algorithms, and artificial neural nets.

## 9. NEXT STEPS

*This section includes tentative plans outlined by the NASA/GSFC IRC team; these plans are highly subject to change.*

The near term efforts of the IRC team fall under two main categories: (1) Enhancing and improving the general IRC framework, including incorporating lessons learned from applying it to SPIRE; and (2) using the IRC framework to develop the instrument control and monitor software for HAWC.

During SPIRE development, we came to the conclusion that it should be easier to develop pipeline algorithms. A major goal is to redesign the pipeline base classes to facilitate the development of custom algorithms without requiring knowledge of the IRC pipeline architecture.

We believe that the Instrument Markup Language can be improved. Currently, it matches the internal model of the instrument fairly closely; it hasn't changed significantly since our initial Java/XML prototyping efforts from late 1998. We think that IML should be optimized for maximum expressiveness and maintainability, and it should model the way instrument designers think when they develop ICDs. For example, IML currently requires the specification of each command in isolation from other commands, and it requires the specification of a command format for each command to be sent to the instrument. However, instrument designers may more naturally group commands into command families, or specify general rules for formatting groups of commands. We plan on redesigning IML to support this and other modeling techniques. We intend to take advantage of the upcoming XML Schema standard which will provide a superset of the capabilities of XMD DTDs.

Currently, the IRC framework only supports a default GUI that enables a user to issue every command and script defined in the IML. This type of GUI is useful for an instrument's engineering test phase, but not as useful for general instrument operations. We always intended to have a way to customize the GUI. Ideas for this involve applying a stylesheet to the IML, which can cause the GUI to be presented differently. Stylesheets could be developed for novice users, expert users, testers, or operators. We even imagine observatory level stylesheets that impose GUI standards across instruments. Our current thinking is to apply emerging XML technology such as Extensible Stylesheet Language (XSL), the Bean Markup Language (BML), or the User Interface Markup Language (UIML), but more research in this area is needed.

In parallel with our efforts to improve the general IRC framework, we will be applying the framework to the development of HAWC's instrument control and monitoring software. Applying the IRC framework to a new instrument involves taking some or all of the following steps:

1. Develop the instrument description. For HAWC, this will involve mapping the ICDs from each of the instrument subsystem teams, including the ICD for SOFIA's telescope assembly, into an IML description for the subsystem.
2. Develop the instrument-specific real-time pipeline algorithms.
3. Develop custom GUIs for the various instrument users.
4. Develop new visualizations.
5. Develop instrument-specific scripts.
6. Develop any instrument-specific delegates (for special purpose parsing or response handling).

## 10. ACKNOWLEDGEMENTS

This work was supported by NASA Goddard Space Flight Center under Contract NAS5-96024. The authors would like to thank the IRC project team: Troy Ames and Carl Hostetter from NASA GSFC; Lisa Koons, Craig Warsaw, Melissa Hess, Steve Clark, Ken Wootton, Ken Sall, Charlie Butterfield from AppNet. Support from the following people has also proven to be invaluable: Dr. Rick Shafer – NASA GSFC SPIRE / SAFIRE team; Dr. Robert Loewenstein and Dr. Rhodri Evans – Yerkes Observatory HAWC team; Dr. Harvey Rhodey – Rochester Institute of Technology HAWC team.

## 11. REFERENCES

1. T. Ames, L. Koons, K. Sall, (1999). *Instrument Remote Control* [Online]. NASA Goddard Space Flight Center. Available: http://pioneer.gsfc.nasa.gov/public/irc/ [2000, February 20]
2. T. Ames, L. Koons, (1999). *IRC Presentations, Publications and Milestones* [Online]. NASA Goddard Space Flight Center. Available: http://pioneer.gsfc.nasa.gov/public/irc/IRC-presentations.html [2000, February 20]
3. T. Ames, L. Koons, K. Sall, (1999). *Instrument Markup Language* [Online]. NASA Goddard Space Flight Center. Available: http://pioneer.gsfc.nasa.gov/public/iml/ [2000, February 20]
4. J. Breed, (1999). *Code 588 Website – Advanced Architectures Automation Branch* [Online]. NASA Goddard Space Flight Center. Available: http://aaaprod.gsfc.nasa.gov/website/WebSite.cfm [2000, February 20]
5. Sterling Software, (1999). *SOFIA Homepage - The latest in Airborne Astronomy* [Online]. NASA Goddard Space Flight Center. Available: http://sofia.arc.nasa.gov/ [2000, February 20]
6. *HAWC* [Online]. Yerkes Observatory. Available: http://astro.uchicago.edu/hawc/hawc.htm [2000, February 20]
7. T. Ames, L. Koons, K. Sall (1999). *Submillimeter And Far Infrared Experiment* [Online]. NASA Goddard Space Flight Center. Available: http://pioneer.gsfc.nasa.gov/public/safire/ [2000 February 20]
8. T. Ames, L. Koons, K. Sall (1999). *Spectral and Photometric Imaging REceiver* [Online]. NASA Goddard Space Flight Center. Available: http://pioneer.gsfc.nasa.gov/public/spire/ [2000 February 20]
9. *The Source for Java™ Technology* [Online]. Sun Microsystems. Available: http://java.sun.com/ [2000 February 20]
10. D. Connolly (2000). *Extensible Markup Language (XML)* [Online]. World Wide Web Consortium. Available: http://www.w3.org/XML/ [2000 February 22]
11. J. Hosler, M. Brandt, K. Hughes (2000). *Advanced Visual Tools and Architectures* [Online]. NASA Goddard Space Flight Center. Available: http://wave.gsfc.nasa.gov/avatar/ [2000 February 20]
12. *JPython Home* [Online]. The Python Consortium. Available: http://www.jpython.org/ [2000 February 20]
13. C. Lilly, V. Quint (2000). *Extensible Stylesheet Language (XSL)* [Online]. World Wide Web Consortium. Available: http://www.w3.org/Style/XSL/ [2000 February 22]
14. S. Weerawarana, M. Duftler (1999). *Bean Markup Language (BML)* [Online]. IBM alphaWorks. Available: http://www.alphaworks.ibm.com/formula/bml [2000 February 20]
15. *UIML.org* [Online]. Virginia Polytechnic Institute. Available: http://www.uiml.org/ [2000 February 20]